Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 6 – Windowed Tables and Window Functions in SQL

Outline

Windowed Table Functions

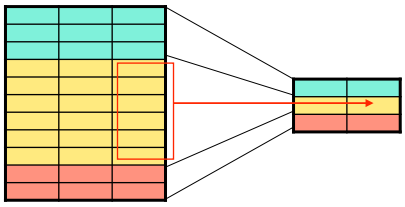
- Windowed table function
  - operates on a window of a table
  - returns a value for every row in that window
  - the value is calculated by taking into consideration values from the set of rows in that window
- 5 new windowed table functions
  - RANK () OVER ...
  - DENSE_RANK () OVER ...
  - PERCENT_RANK () OVER ...
  - CUME_DIST () OVER ...
  - ROW_NUMBER () OVER ...
- In addition, 8 old aggregate functions and 16 new aggregate functions can also be used as windowed table functions:

  Example: sum(salary) OVER ...
- Allows calculation of moving and cumulative aggregate values.

Concept (Compared To Set Functions)

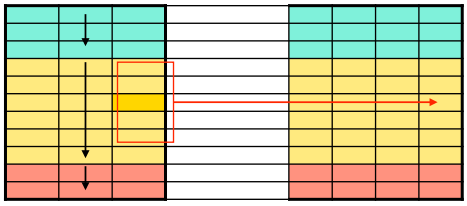
- Set functions (aggregate functions)

```
SELECT dept, AVG(salary)
FROM Employees
GROUP BY dept
```



- Windowed Table Functions (tuple-based aggregation)

```
SELECT dept, empno, salary,
  AVG(salary) OVER(
      PARTITION BY dept
      ORDER BY age
      ROWS
        BETWEEN 2 PRECEEDING
            AND 2 FOLLOWING)
FROM Employees
```

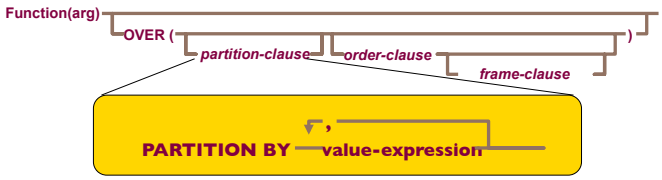


Windowed Tables and Window Functions

- Windowed table
  - table (result of a table expression) together with one or more windows
  - windows are independent from each other
- Window
  - defines, for each row in the table, a set of rows (*current row window*) that is used to compute additional attributes
  - specified using a window specification (OVER clause)
  - based on three main concepts
    - window partitioning is similar to forming groups, but rows are retained
    - window ordering defines an order (sequence) of rows within each partition
    - window frame is defined relative to each row to further restrict the set of rows
- Window function
  - is applied for each row, over the current row window, returning a single value
  - used in column expressions in the select-list

The Partitioning Clause

- The partition-clause allows to subdivide the rows into partitions, much like the group by clause



- Without further clauses, the current row window contains all the rows of the same partition (i.e., all the rows that are not distinct from the current row, including the current row)
  - if no partitioning clause is specified, then there is a single partition that contains the complete table
- Windows do not reach across partition boundaries!

Set Functions as Window Functions

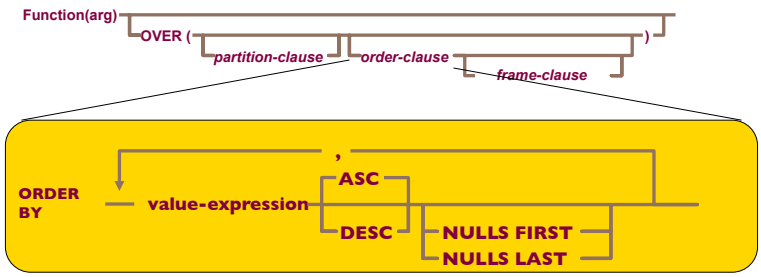
- The OVER clause turns a set function into a window function
  - Aggregated value is computed per current row window (here: per partition)

```
select empnum, dept, salary,
       avg(salary) over (partition by dept) as dept_avg
from emptab;
```

EMPNUM	DEPT	SALARY	DEPT_AVG
6	1	78000	63833
2	1	75000	63833
7	1	75000	63833
11	1	53000	63833
5	1	52000	63833
1	1	50000	63833
9	2	51000	51000
4	2	-	51000
8	3	79000	69667
12	3	75000	69667
10	3	55000	69667
3	-	84000	84000
0	-	-	84000

The Order Clause

- The order-clause defines an order (sequence) within a partition
- May contain multiple order items
  - Each item includes a value-expression
  - NULLS FIRST/LAST defines ordering semantics for NULL values
- This clause is completely independent of the query's ORDER BY clause



Ranking Functions For Sequences

- RANK
  - returns the relative position of a value in an ordered group
  - equal values (ties) are ranked the same
- DENSE_RANK
  - like RANK, but no gaps in rankings in the case of ties
- ROW_NUMBER
  - ties are non-deterministically numbered
- Ordering is required!
- Example:

```
select empnum, dept, salary,
       rank() over (order by salary desc nulls last) as rank,
       dense_rank() over (order by salary desc nulls last) as denserank,
       row_number() over (order by salary desc nulls last) as rownum
from emptab;
```

Ranking Functions Example

EMPNUM	DEPT	SALARY	RANK	DENSERANK	ROWNUM
3	-	84000	1	1	1
8	3	79000	2	2	2
6	1	78000	3	3	3
2	1	**75000**	**4**	**4**	**4**
7	1	**75000**	**4**	**4**	**5**
12	3	**75000**	**4**	**4**	**6**
10	3	55000	**7**	**5**	**7**
11	1	53000	8	6	8
5	1	52000	9	7	9
9	2	51000	10	8	10
1	1	50000	11	9	11
4	2	-	**12**	**10**	**12**
0	-	-	**12**	**10**	**13**

Example: Rank with Ordering and Partitioning

- Find rankings of each employee's salary within her department

```
select empnum, dept, salary,
  rank() over (partition by dept order by salary desc nulls last)
        as rank_in_dept,
  rank() over (order by salary desc nulls last) as globalrank
from emptab;
```

EMPNUM	DEPT	SALARY	RANK_IN_DEPT	RANK
6	1	78000	1	3
2	1	75000	2	4
7	1	75000	2	4
11	1	53000	4	8
5	1	52000	5	9
1	1	50000	6	11
9	2	51000	1	10
4	2	-	2	12
8	3	79000	1	2
12	3	75000	2	4
10	3	55000	3	7
3	-	84000	1	1
0	-	-	2	12

Rank on Aggregations

- Windowed table functions are computed in the select list
  - After applying FROM, WHERE, GROUP BY, HAVING
  - They may not be referenced in any of these clauses
  - May use aggregation functions in window specification expressions
  - If you wish to reference them, you must nest them, or use a common table expression
- Example: Find rankings of each department's total salary

```
select dept, sum(salary) as sumsal,
     rank() over (order by sum(salary) desc nulls last) as rankdept
from emptab
group by dept;
```

DEPT	SUMSAL	RANKDEPT
1	383000	1
3	209000	2
-	84000	3
2	51000	4

Cumulative Functions with Partitioning

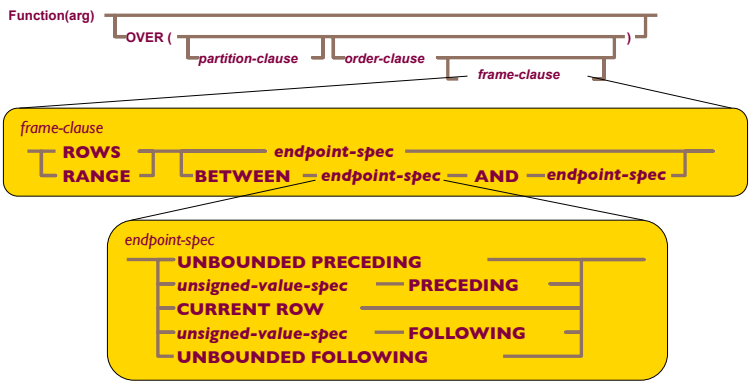
- Without a frame-clause, the current row window is now restricted to all rows equal to or preceding the current row within the current partition
  - Example: *Find the total sales per quarter, and cumulative sales in quarter order PER YEAR for 1993-1995*

```
select year, quarter, sum(s.dollars) as q_sales,
    sum(sum(s.dollars)) over
    (partition by year
    order by quarter)
      as cume_sales_year
from sales s
where year between 1993 and 1995
group by year, quarter;
```

YEAR	QUARTER	Q_SALES	CUME_SALES_YEAR
1993	1	1270775.75	1270775.75
1993	2	1279171.45	2549947.20
1993	3	1050825.44	3600772.64
1993	4	1062329.99	4663102.63
1994	1	1176312.84	1176312.84
1994	2	1132602.73	2308915.57
1994	3	1241437.72	3550353.29
1994	4	1103020.49	4653373.78
1995	1	1193343.62	1193343.62
1995	2	1194296.14	2387639.76
1995	3	1418400.68	3806040.44
1995	4	1182153.01	4988193.45

Window Frames

- Further refine the set of rows in a function's window when an order by is present
  - Allows inclusion/exclusion of ranges of values or rows within the ordering



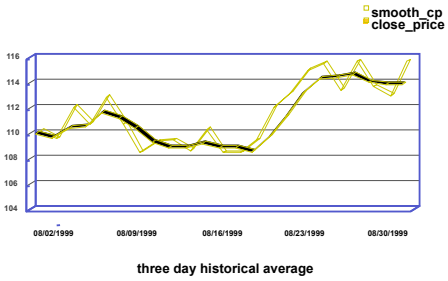
Example: Curve Smoothing

Find the three day historical average of IBM stock for each day it traded

```
select date,symbol, close_price,
avg(close_price) over (order by date rows 2 preceding) as
 smooth_cp
from stocktab
where symbol = 'IBM' and date between '1999-08-01' and '1999-09-01';
```

DATE	SYMBOL	CLOSE_PRICE	SMOOTH_CP
08/02/1999	IBM	110.125	110.1250
08/03/1999	IBM	109.500	109.8125
08/04/1999	IBM	112.000	110.5416
08/05/1999	IBM	110.625	110.7083
08/06/1999	IBM	112.750	111.7916
08/09/1999	IBM	110.625	111.3333
08/10/1999	IBM	108.375	110.5833
08/11/1999	IBM	109.250	109.4166
08/12/1999	IBM	109.375	109.0000
08/13/1999	IBM	108.500	109.0416
08/16/1999	IBM	110.250	109.3750
08/17/1999	IBM	108.375	109.0416
08/18/1999	IBM	108.375	109.0000
08/19/1999	IBM	109.375	108.7083
08/20/1999	IBM	112.000	109.9166
08/23/1999	IBM	113.125	111.5000
08/24/1999	IBM	114.875	113.3333
08/25/1999	IBM	115.500	114.5000
08/26/1999	IBM	113.375	114.5833
08/27/1999	IBM	115.625	114.8333

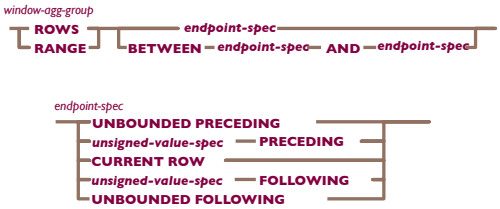
- Now the curve is smooth, but it is uncentered
- Centered average:
  - ... rows between 1 preceding and 1 following ...



three day historical average

RANGE Based Windows

DATE	SYMBOL	CLOSE_PRICE
08/02/1999	IBM	110.125
08/03/1999	IBM	109.500
08/04/1999	IBM	112.000
08/05/1999	IBM	110.625
08/06/1999	IBM	112.750
values missing for the weekend!		
08/09/1999	IBM	110.625
08/10/1999	IBM	108.375
08/11/1999	IBM	109.250
08/12/1999	IBM	109.375
08/13/1999	IBM	108.500
.. and here		
08/16/1999	IBM	110.250



- ROW based windows work great when the data is dense
  - duplicate values and missing rows can cause problems
- In other situations, it would be nice to specify the aggregation group in terms of values, not absolute row position
  - For example, the stock table doesn't have any entries for weekends
  - Looking at the last 6 rows gives you more than the last week

RANGE Based Window Example

For IBM stock, what is the 7 calendar day historical average, and the 7 trade day historical average for each day in the month of August, 1999

```
select date,substr(dayname(date),1,9), close_price,
       avg(close_price) over (order by date rows 6 preceding) as avg_7_rows,
       count(close_price) over (order by date rows 6 preceding) as count_7_rows,
       avg(close_price) over (order by date range interval '6' day preceding) as avg_7_range,
       count(close_price) over (order by date range interval '6' day preceding) as count_7_range
from stocktab
where symbol = 'IBM' and date between '1999-08-01' and '1999-09-01';
```

```
DATE       2         CLOSE_PRICE  AVG_7_ROWS COUNT_7_ROWS AVG_7_RANGE COUNT_7_RANGE
---------- --------- ------------ ---------- ------------ ----------- -------------
08/02/1999 Monday          110.125     110.12            1      110.12             1
08/03/1999 Tuesday         109.500     109.81            2      109.81             2
08/04/1999 Wednesday       112.000     110.54            3      110.54             3
08/05/1999 Thursday        110.625     110.56            4      110.56             4
08/06/1999 Friday          112.750     111.00            5      111.00             5
08/09/1999 Monday          110.625     110.93            6      111.10             5
08/10/1999 Tuesday         108.375     110.57            7      110.87             5
08/11/1999 Wednesday       109.250     110.44            7      110.32             5
08/12/1999 Thursday        109.375     110.42            7      110.07             5
08/13/1999 Friday          108.500     109.92            7      109.22             5
08/16/1999 Monday          110.250     109.87            7      109.15             5
08/17/1999 Tuesday         108.375     109.25            7      109.15             5
...
```

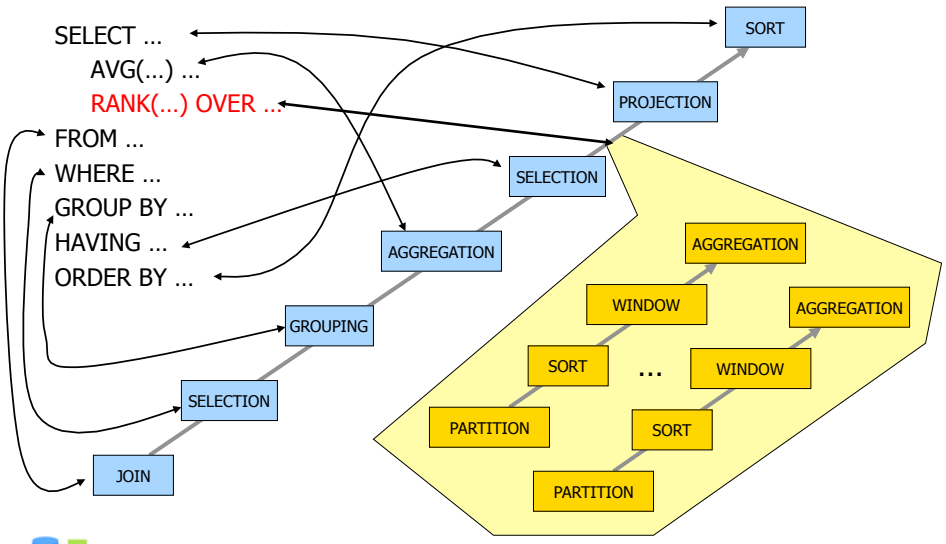
Explicit Window Definition Clause

- So far, a window was specified "in-line" in the SELECT clause of a query
- Alternative syntax uses an explicit WINDOW clause

```
select date, symbol, close_price,
avg(close_price) over w as smooth_cp
from stocktab
where symbol = 'IBM' and date between '1999-08-01' and '1999-09-01'
window w as (order by date rows 2 preceding)
```

- Advantages
  - window has a name, which can be used by multiple window table function invocations in the SELECT clause

SQL Query Processing Steps incl. OLAP



Additional Capabilities

- Hypothetical Aggregate Functions
  - 4 new hypothetical aggregate functions:
    - RANK (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
    - DENSE_RANK (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
    - PERCENT_RANK (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
    - CUME_DIST (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
  - Hypothetical aggregate functions evaluate the aggregate over the window extended with a new row derived from the specified values.
    - "What if" scenarios
- Inverse Distribution Functions
  - 2 new inverse distribution functions:
    - PERCENTILE_DISC (expr) WITHIN GROUP (ORDER BY <sort specification list>)
    - PERCENTILE_CONT (expr) WITHIN GROUP (ORDER BY <sort specification list>)
  - Argument must evaluate to a value between 0 and 1.
  - Return the values of expressions specified in <sort specification list> that correspond to the specified percentile value.

SQL:2003 Built-in Functions for OLAP

- 34 new built-in functions:
  - 7 new numeric functions
  - 16 new aggregate functions
  - 5 new windowed table functions
  - 4 new hypothetical aggregate functions
  - 2 new inverse distribution functions
- Windowed table functions provide facilities for calculating moving sums, moving averages, ranks, correlation, standard deviation, regression, etc.
- Significant functionality and performance advantages for OLAP applications

New Built-in Functions

- 7 new numeric functions
  - LN (expr)
  - EXP (expr)
  - POWER (expr, expr)
  - SQRT (expr)
  - FLOOR (expr)
  - CEIL[ING] (expr)
  - WIDTH_BUCKET(expr, expr, expr, expr)
    EX: WIDTH_BUCKET (age, 0, 100, 10)
- 16 new aggregate functions
  - STDDEV_POP (expr)
  - STDDEV_SAMP (expr)
  - VAR_POP (expr)
  - VAR_SAMP (expr)
  - COVAR_POP (expr, expr)
  - COVAR_SAMP (expr, expr)
  - CORR (expr, expr)
  - REGR_SLOPE (expr, expr)
  - REGR_INTERCEPT (expr, expr)
  - REGR_COUNT (expr, expr)
  - REGR_R2 (expr, expr)
  - REGR_AVGX (expr, expr)
  - REGR_AVGY (expr, expr)
  - REGR_SXX (expr, expr)
  - REGR_SYY (expr, expr)
  - REGR_SXY (expr, expr)

Summary

- OLAP-Functionality in SQL
  - extension of classical application of aggregation functions
- Windowed tables, window functions
  - tuple-based, attribute-based partitioning and analysis/aggregation of data
  - rows in a partition are preserved/expanded
    - in contrast to group-by/aggregation
  - window order defines sequence for sequence-based analysis
    - cumulative aggregation, ranking
  - window frame defines current row window dynamically for ordered windows
    - moving aggregates
- Multiple windows can be defined for the same table
  - windows are independent
- SQL query execution model enhancement
- This functionality provides powerful infrastructure for optimized data analysis in the scope of OLAP